Machine Learning in Ultra-High Energy (UHE) Neutrino Analysis

Abdullah Alhag with Professor Amy Connolly
July 2019

Abstract

The Antarctic Impulsive Transient Antenna (ANITA) is a NASA long-duration balloon experiment with the primary goal of detecting ultra-high-energy (> 10^{17} eV) neutrinos via the Askaryan Effect. This research investigates the usability of a Convolution Neural Network (CNN), a form of machine learning, in differentiating a form of background noise from the data obtained by ANITA from other types of signals. The background noise events of interest here are "payload blasts," which are background noise events caused by an unknown object on the ANITA payload. CNN is a technique most commonly used in analyzing visual imagery. It is built on the idea of multilayer perceptron, which is used in classifying nonlinear data. The classification is done by identifying features that are special to the set of events being classified. Both TensorFlow [1] and PyTorch [2] were used to create models that can classify the payload blasts from ANITA data vs. non-payload events. These models however can be extended to classify other events that are of interest. The trained CNN models were able to accurately classify the payload blasts with most models being able to achieve an accuracy of around 98%.

Acknowledgments

Thanks to Professor Amy Connolly for mentoring me through this research project. Thanks to everyone in my research group at The Ohio State, in particular, Oindree Banerjee, who was the first to start working on the project. I would also to thank Carl Pfendner and Brian Clark who were there for me when I needed help. Many thanks to the College of Engineering for the financial support.

Table of Contents

List of Figures

1.0. Introduction to Ultra High Energy (UHE) Neutrinos

1.1. What Are UHE Neutrinos?

Neutrinos are fermions with no charge and negligible mass. UHE neutrinos have greater than approximately 10^{17} eV in energy and are theorized to exist; when cosmic rays interact with the cosmic microwave background, they produce UHE neutrinos in a phenomenon known as the Greisen–Zatsepin–Kuzmin interaction [3]. When neutrinos interact in a medium, they produce a particle bunch that propagates through the medium; a description of how ANITA sees a neutrino is depicted in Figure 1. The particle bunch gives off electromagnetic radiation in a phenomenon known as the Askaryan effect [4], which is coherent for wavelengths greater than the 10-cm bunch size, equivalent to the light frequency of around under 1 GHz. The interest in UHE neutrinos comes from being uncharged and therefore only interact weakly as they travel across the universe. This means that when neutrinos arrive at the earth, they carry information about the direction of the source that created the particles.

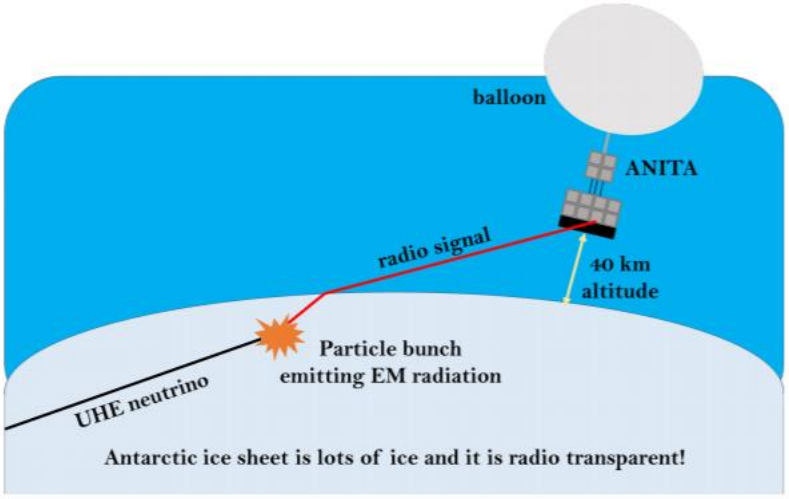

Figure 1: *Detection of UHE neutrinos using ANITA [5].*

1.2. The Antarctica Impulsive Transient Antenna (ANITA)

ANITA is a radio detector suspended by a balloon in Antarctica, which surveys a large area of ice and records potential neutrino events, shown in Figure 2. Thus far, 4 flights, each about one month long, have launched. The ANITA payloads consist of its radio frequency antennas and signal processing units. The payloads on these flights are predicted to produce a noise that interferes with our measurement, hence the name payload blast. The task is to differentiate between Neutrino events and background events, the background events here being the payload blasts. Figure 3 shows the waveform differences between payload blasts vs. non-payload events.

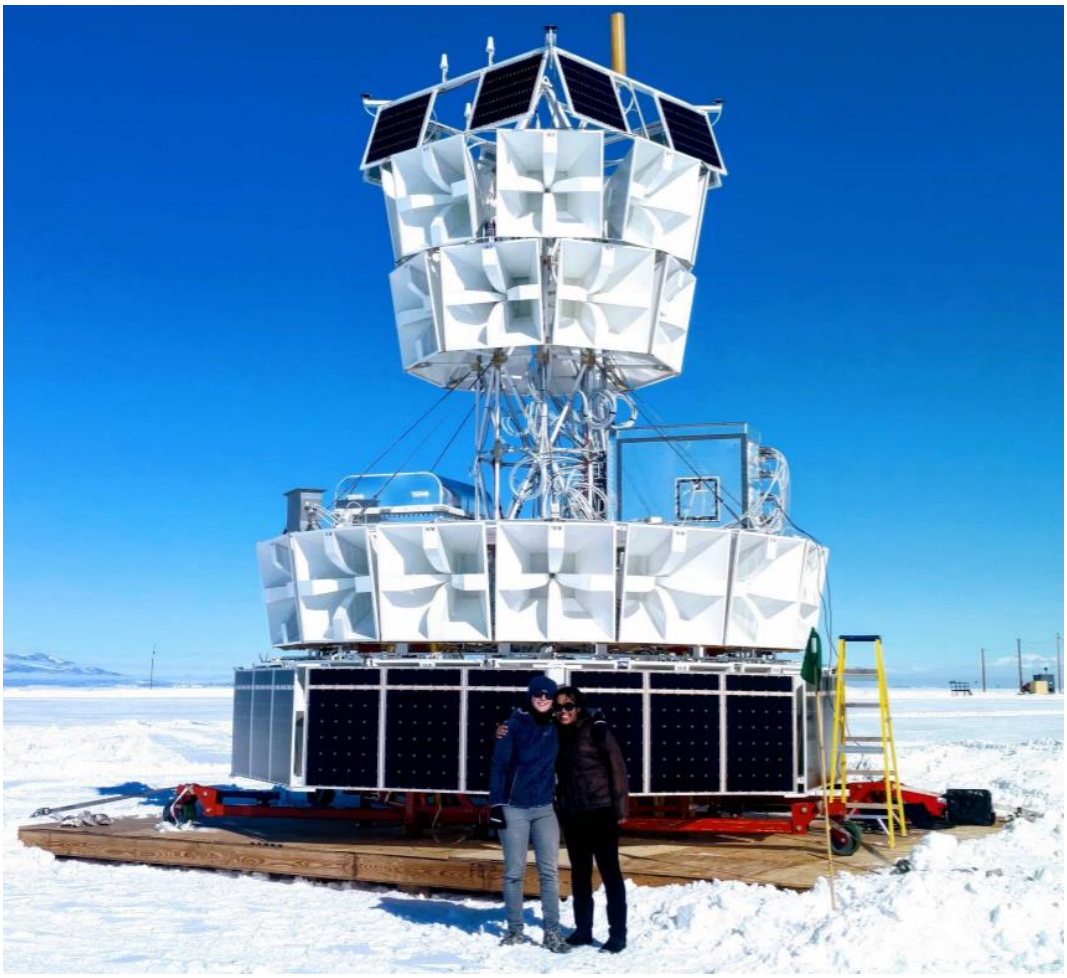

Figure 2: *ANITA-4 before its launch, McMurdo Station, Antarctica [5]*

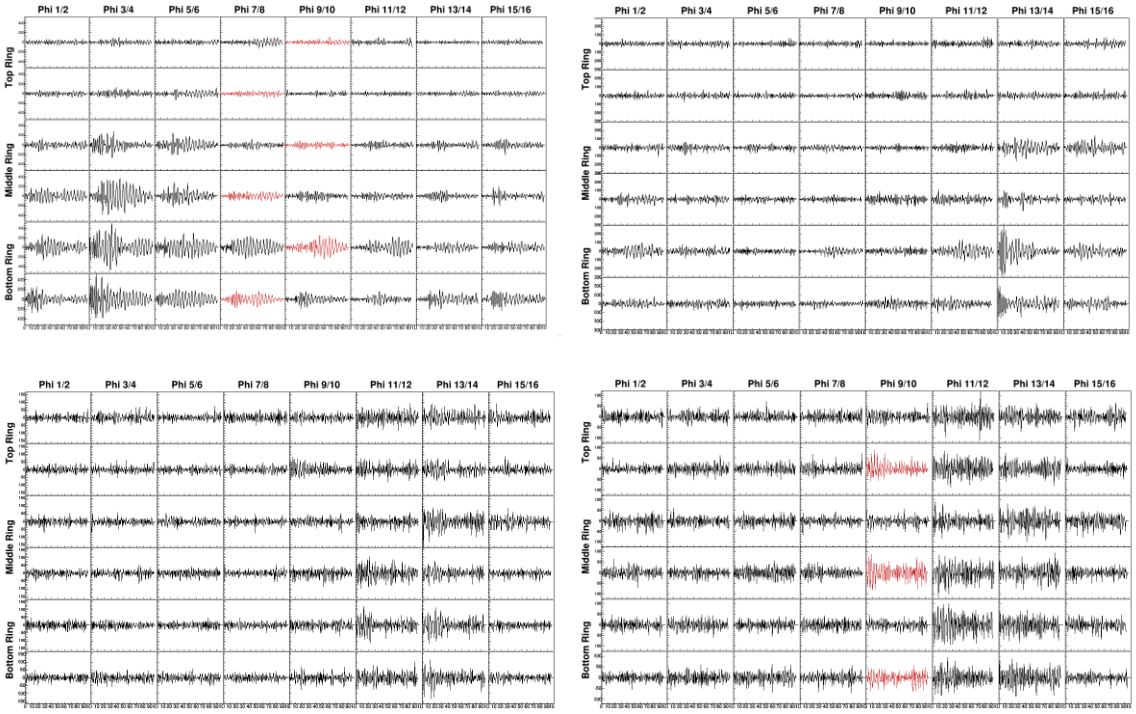

Figure 3: *Payload (Top) vs. non-payload events (Bottom)*

2.0. Previous Analysis Work

A previous member of the group, Dr. Oindree Banerjee, developed a project called Blastfamy with the goal of finding payload blasts in the ANITA dataset. Her approach was to use Principal Component Analysis (PCA) using 10 features [5]. Figure 4 shows that the first three principal components are the most important ones as the variances associated with each one falls off exponentially. PCA is mainly used to reduce the dimensionality of a multi-dimensional dataset. This dimensionality reduction is accomplished by calculating linear combinations from the features and determining which linear combinations can describe the data the best. Her findings indicate that this method, as it is now, was not able to reject payload blasts; instead, it was able to reject low-quality events such as digitizer glitches. Another proposed method in her thesis was to use more features or a semi-supervised method such as a Linear Discriminant Analysis. In this work, we instead investigate the use of machine learning in the form of CNN.

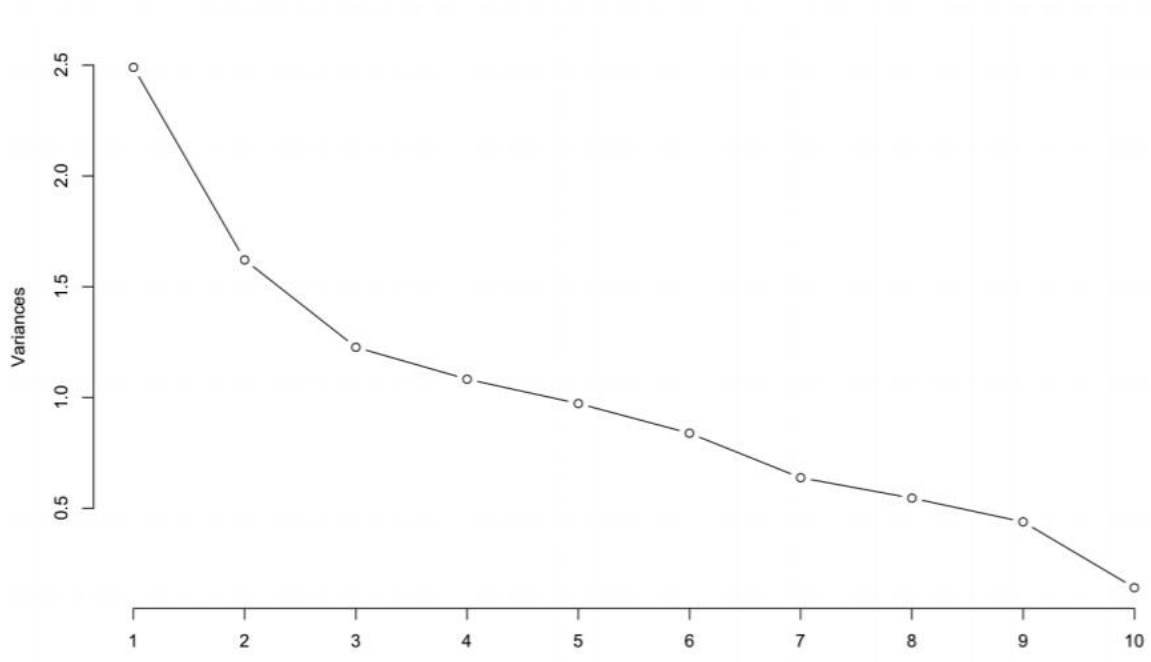

Figure 4: *Principal components and variances using 10 features in PCA [5].*

3.0. Introduction to Convolution Neural Network (CNN)

3.1. Motivation Behind CNN

The data collected by ANITA exhibit locality; the waveform of payload blasts is significantly different from that of other events, and CNNs are well known for their ability in capturing local features. This makes CNNs very good at classifying images where the location of a collection of pixels represents a feature, a dog nose for example. Similarly, with sound, the amplitude of a sound wave in a time domain represents a feature. Therefore, solving the problem of classifying payload blasts using CNN makes sense as the waveforms collected by ANITA exhibit features locality.

3.2. The Working of CNN

CNN is a proven method for classifying images of different classes with accuracy approaching 100% given enough training data. For those interested in learning more about CNN,

the book Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville includes topics that go beyond what is mentioned here [8]. In attempting to explain CNN, it is best to see how it works on two simple examples and explain how we can generalize these examples. The first example is of 2 by 2 pixels input where only 2 characters are allowed, \ and /, both represented as an array as shown in Figure 5.

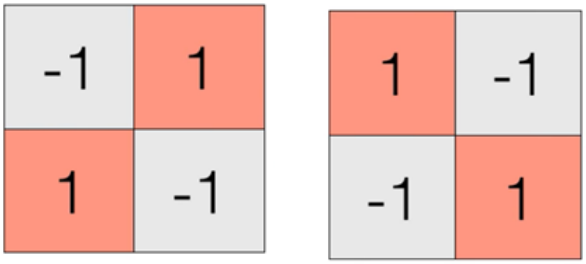

Figure 5: Visual r*epresentation of the / and \ characters [6]*.

In attempting to classify these two characters, CNN will try to guess filters that can distinguish the two characters. The possibilities for such guesses are unlimited. For the example above however, it is enough to only consider filters where the only allowed parameters are +1(+) and -1(-) and therefore find that there are 16 possible solutions as shown in Figure 6.

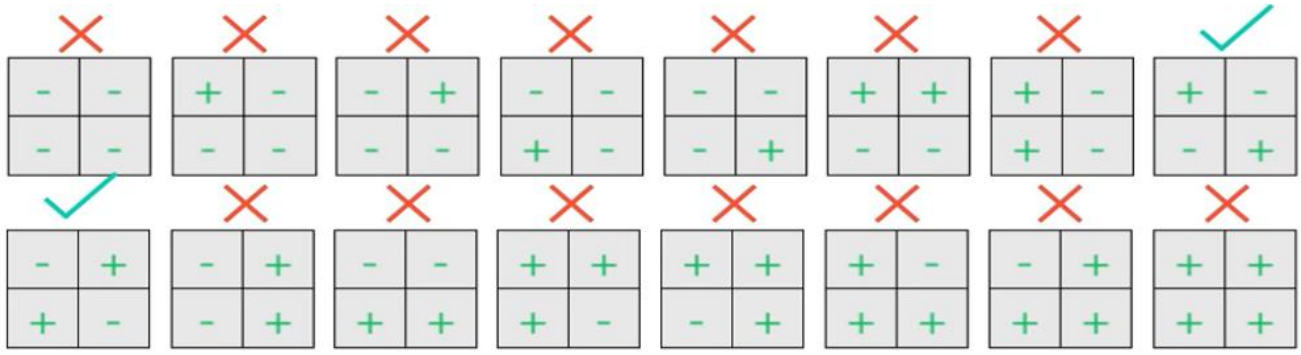

Figure 6: *The 16 possible filters the algorithm needs to test [6]*.

It can be seen that from the 16 possible filters, only two can actually be used to classify the characters correctly. This can be verified by picking one of the correct filters and applying the convolution operator on the character array, namely multiplying the elements of the array by the

corresponding +/- and adding all the elements of the array as shown in Figure 7.

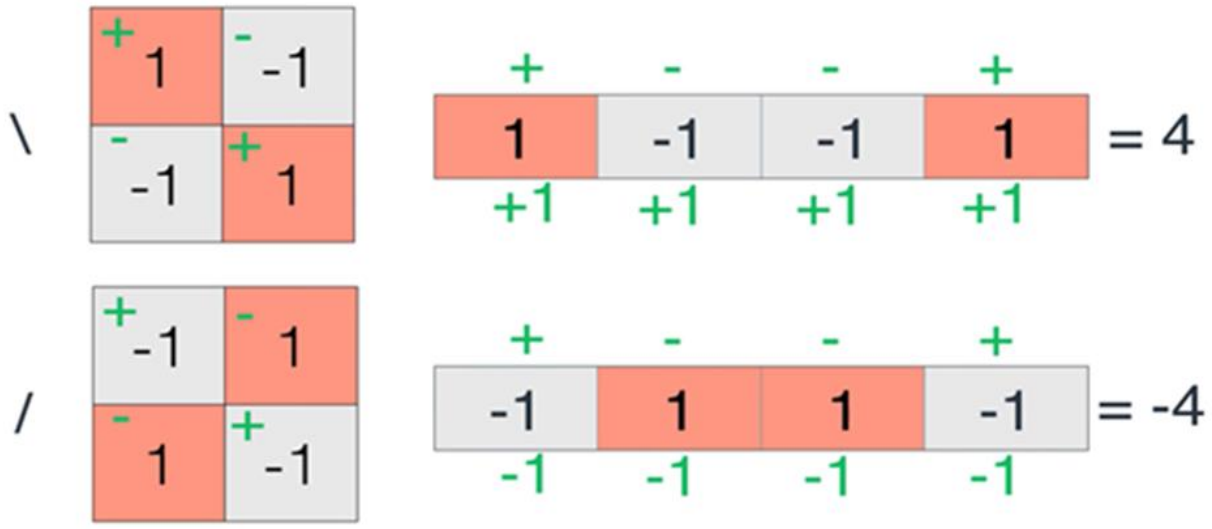

Figure 7: *Classifying / and \ characters using one of the models [6].*

It is worth pointing out that the process of finding filters is not done by testing all possible solutions; it is rather done by gradient descent. The algorithm starts with a random guess generation and tries to evolve it by making small changes on the best models of each generation until a solution is found, Figure 8.

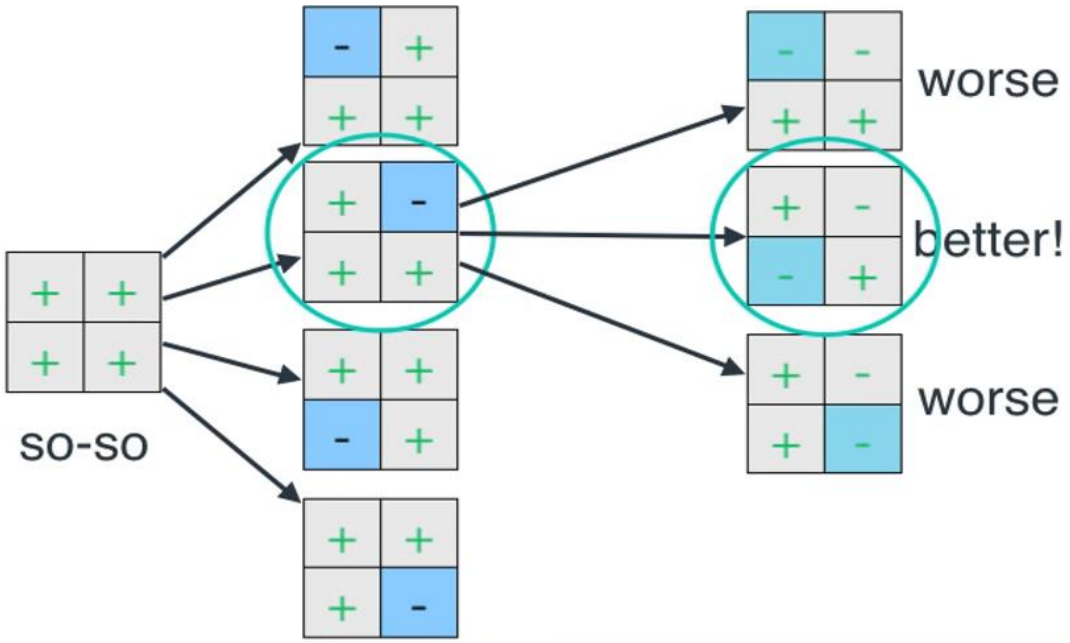

Figure 8: *Gradient descent for evolving the mathematical models [6]*

The filters we used in the previous example are referred to as Convolution Layers. They are the basic block of any CNN and are responsible for extracting features. To build a Convolution Layer, a few parameters need to be chosen: the window size and the stride size. The window size in the previous example was 2 by 2; a complete model would usually implement 3-dimensional Convolution Layers, where the first 2 dimensions capture the spatial features and the 3rd dimension captures other features such as colors. The stride size, on the other hand, refers to how the filters are slid over the image.

Another layer that is used to construct a CNN is a Pooling Layer. The most commonly used Pooling Layer is a Max Pooling Layer. This layer functions to down-sample an input representation and therefore reduce the number of parameters, see Figure 9. Similar to the Convolution Layer, the window size and the stride size of the Pooling Layers are parameters chosen by the implementer when building a model. The last Layer of any CNN is a Fully Connected Layer. This layer is composed of neurons that have full connections to all activations in the previous layer. Each neuron receives an input, performs a transformation on the input, and outputs a single scalar value. The transformation is an activation function. In the previous example, the activation function is simply an if statement whose outcomes depends on whether the output is greater or less than 0. If less than 0, the character is determined to be a forward slash; if otherwise, it is determined to be a backward slash.

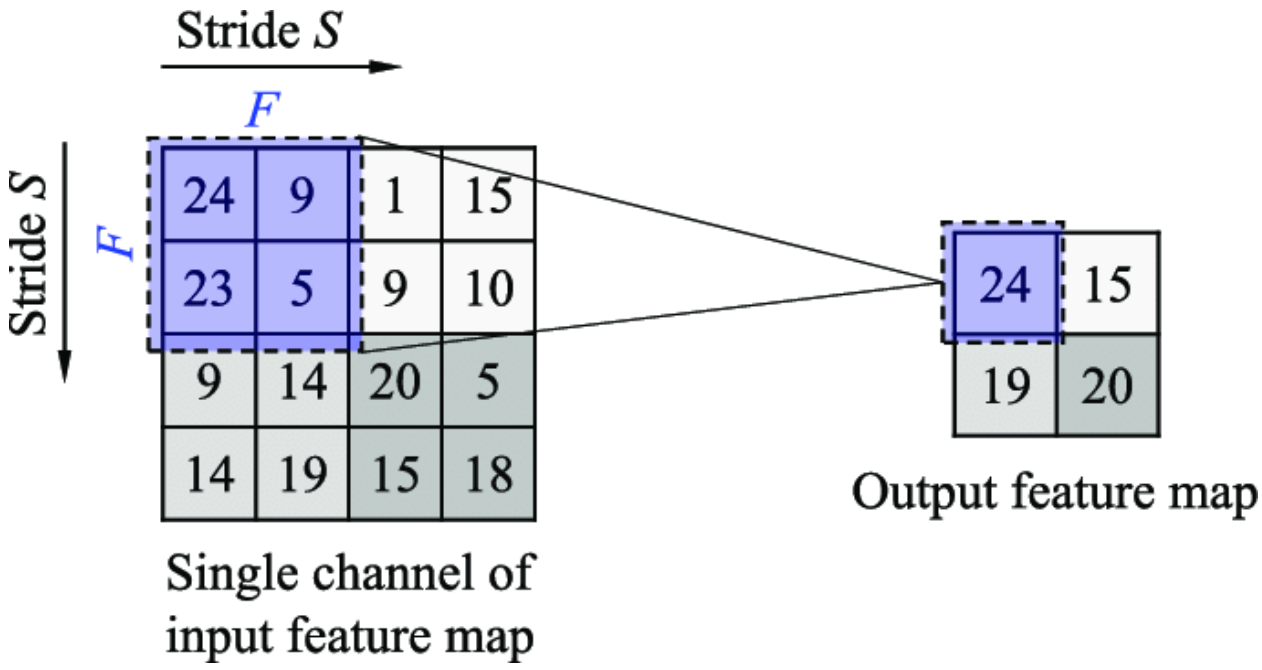

Figure 9: *An example showing how a max pooling layer reduces the special complexity of a 4 by 4 array to a 2 by 2 array [7]*

In the next example, we look into a more complex input to see how the model complexity scales and to put the previously explained layers to use. In this example, we look into a 3*3 pixels input. For simplicity, we limit the number of characters represented by the 3*3 pixels to 4, in particular, \, /, X, and O. To reduce the number of parameters that need training we use decoding, a process of representing images in a simple form and, hence reducing the complexity. For this example, it is sufficient to describe the 3*3 pixels characters in terms of the previous 2*2 pixels character which is equivalent to applying a Convolution Layer and a Pooling Layer. Next thing to do is to use the filters from the previous layers and based on an activation function we can determine the class an input belongs to; this is equivalent to applying a Fully Connected Layer. These steps are described in Figure 10.

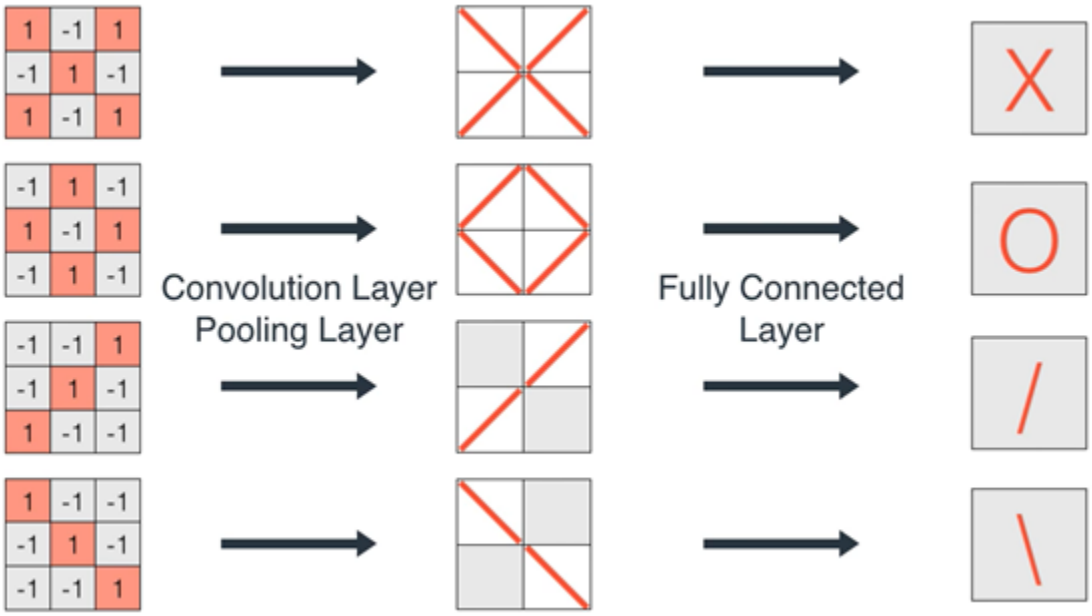

Figure 10: *A summary of the 3*3 pixels model [6].*

To see how these different layers help classify the different characters, it is convenient to show their action on the X character. The algorithm starts by randomly initializing all the layers and it only stops changing the parameters, when the accuracy determined by the user is achieved, optimally 100%. Assuming this evolution process was successful, the final filters of the Convolution Layer might look similar to what is shown in Figure 11.

Figure 11: *Backward and forward slash filters [6].*

Using these filters, our algorithm can determine that the X character is nothing more than a combination of a forward and a backward slash. To see this, we apply the convolution operator

on the X character in a stride size of 1. This means that we multiply the elements of the input array by the corresponding signs of the filters and adding. The result of the convolution operator is shown in Figure 12.

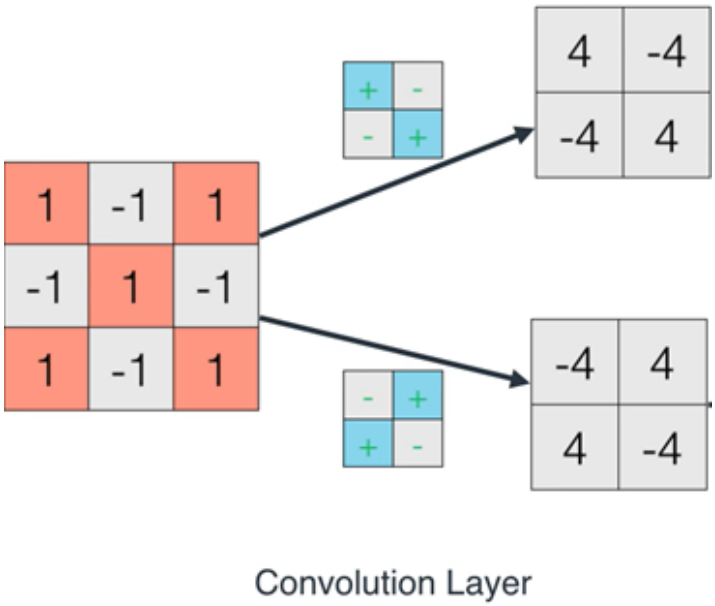

Figure 12: *Results of applying a Convolution Layer [6]*.

We can also utilize a Max Pooling Layer for this example. This layer only keeps the maximum values of the matrix that results from applying the convolution operator on the input. In the case of the array resulting from the forward slash filter, only the right top and bottom left values are kept since both values are 4. Figure 13 shows the action of applying the Max Pooling Layer.

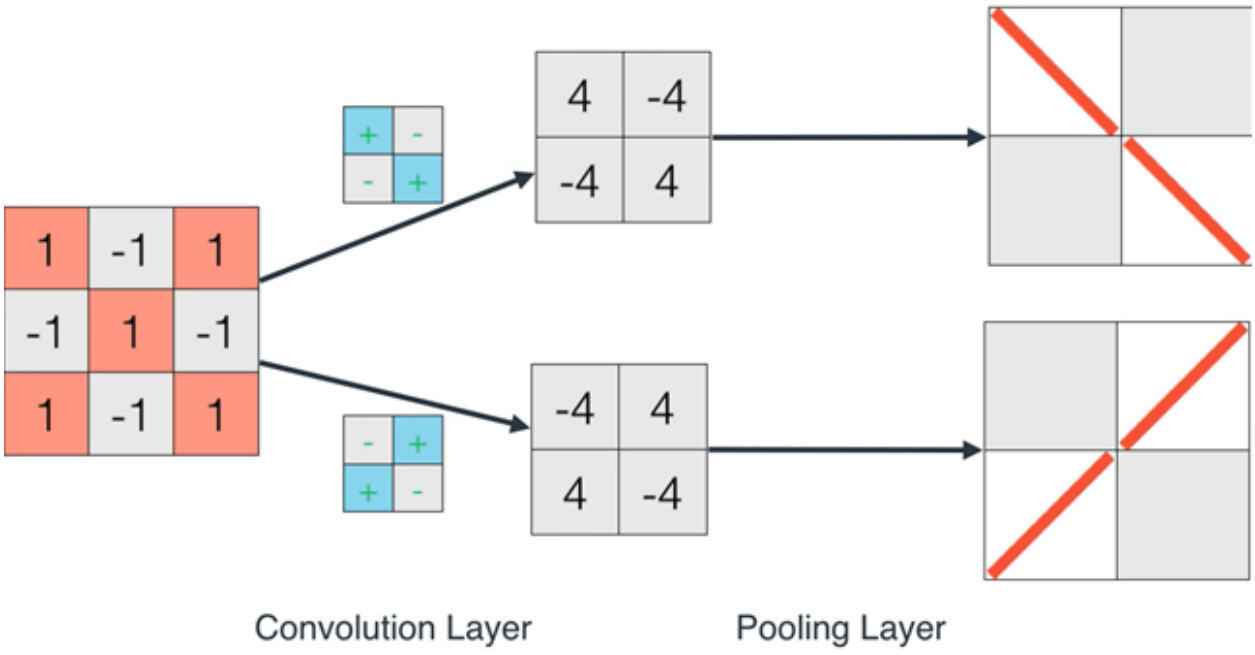

Figure 13: *Results of applying a Convolution Layer and a Max Pooling Layer [6].*

In summary, the Convolution Layer and the Pooling Layer reduces the complexity of the X character, which can be seen by over-imposing the two results, represented in Figure 14.

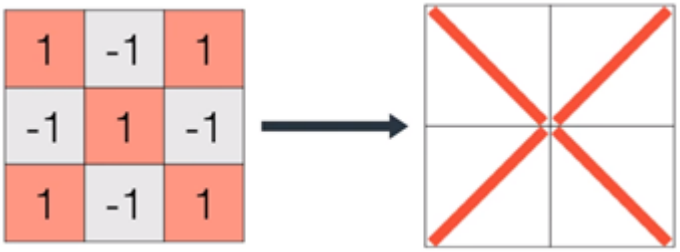

Figure 14: *X Character representation after applying a Convolution Layer and a Max Pooling Layer [6].*

By applying the Convolution Layer and the Pooling Layer on the other characters, we end up with a 2 by 2 pixels representation for all 4 characters. We could also simplify things further by decoding the 4 inputs to a 1-d array. This is conveniently shown in Figure 15.

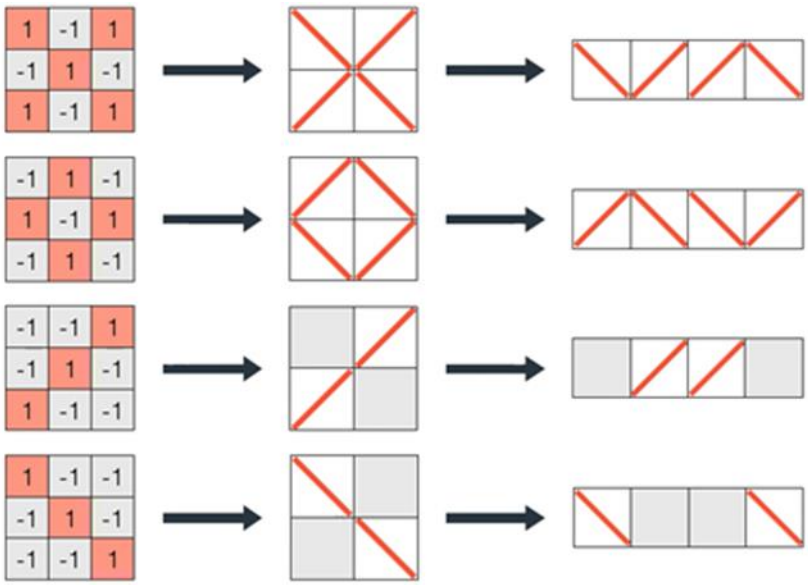

Figure 15: *The final representation of all 4 characters [6].*

For this example, it is also convenient to encode the character as a matrix in / and \ vector space. This is equivalent to passing / and \ filters over the 1-d reduced input image and assigning the values 1 (same) and -1 (different) to the elements of the matrix, resulting in what is shown in Figure 16.

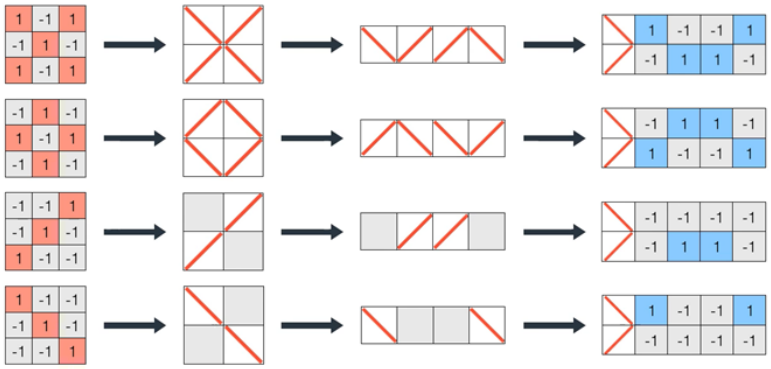

Figure 16: *The representation of the characters in / and \ vector space [6].*

At this point, we could build our final filters based on the representation of the characters in the / and \ vector space. This is done by mapping 1 to a plus (+) and -1 to a minus (-) as shown in Figure 17.

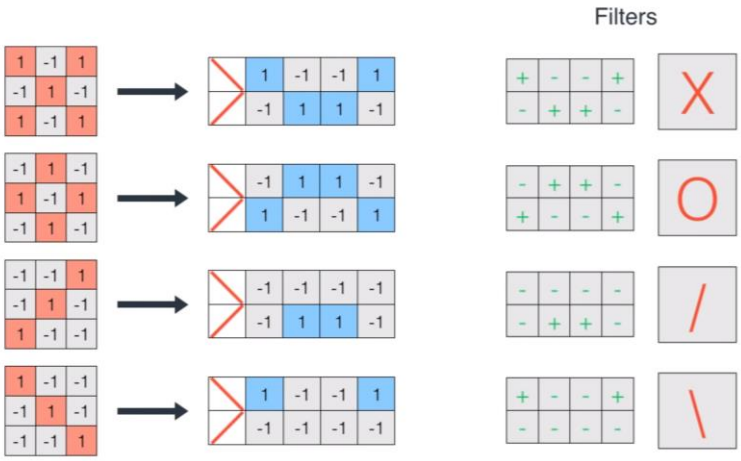

Figure 17: *Final filters used to classify the characters [6].*

From this, we observe that each filter will give the highest score only when applied to the character it is built from. Therefore, only when passing the correct filter over the input character, we will get a high score, a neuron activation. This is best shown by working out how the algorithm classifies the X character as shown in Figure 18.

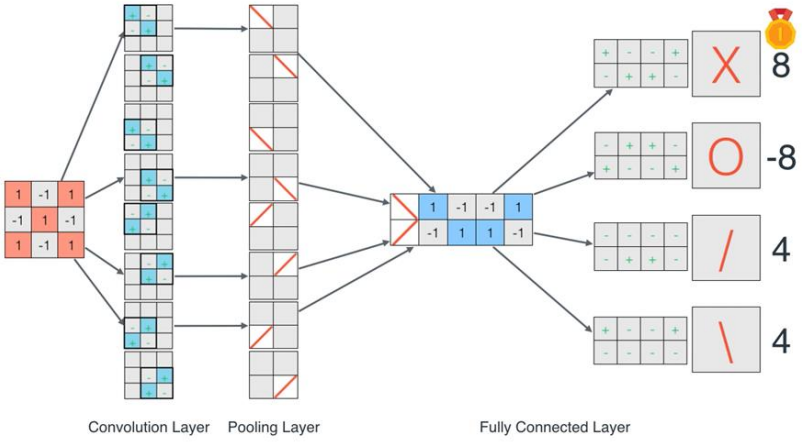

Figure 18: *Classifying the X character by applying the four filters to see how they score [6].*

This example might lead the reader to wonder how evolving filters that match the input can help classify a real image. To see why this is the case, we could work out what a cat-dog classifier might look like. For such a classifier, the filters might be something along the lines of a dog nose, a dog mouth, a cat ear, etc. Therefore, only the filters associated with the input being classified, a picture of a dog or cat, will activate allowing the model to accurately classify the input. Figure 19 shows what a fully functioning model might look like.

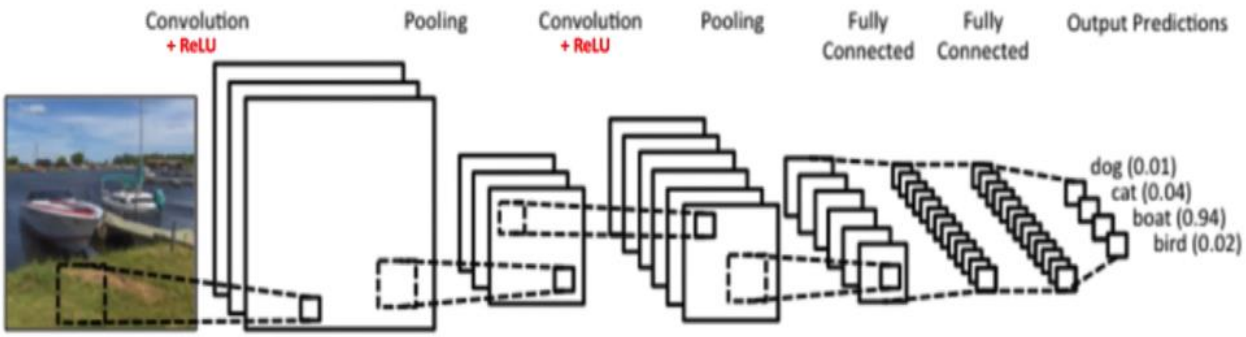

Figure 19: A *Real-world example of applying a CNN [6]*.

3.3. Gradient Descent

One key component to the efficiency of CNNs is the use of gradient descent. Gradient descent is an iterative method that is used to update the model's parameters to make extracting features possible by minimizing a cost function, also known as a loss function. The cost function in CNN is what is used to measure how far our prediction is from the expected. A common loss function is the Mean Squared Error. Therefore, gradient descent is an optimization algorithm that works by minimizing a loss function by moving in the direction of the steepest descent as defined by the negative of the gradient as shown in Figure 20. There are three types of Gradient Descent: Batch Gradient Descent, Stochastic Gradient Descent (SGD), and Mini-batch Gradient Descent. All these differ on how much of the data is selected for each training cycle.

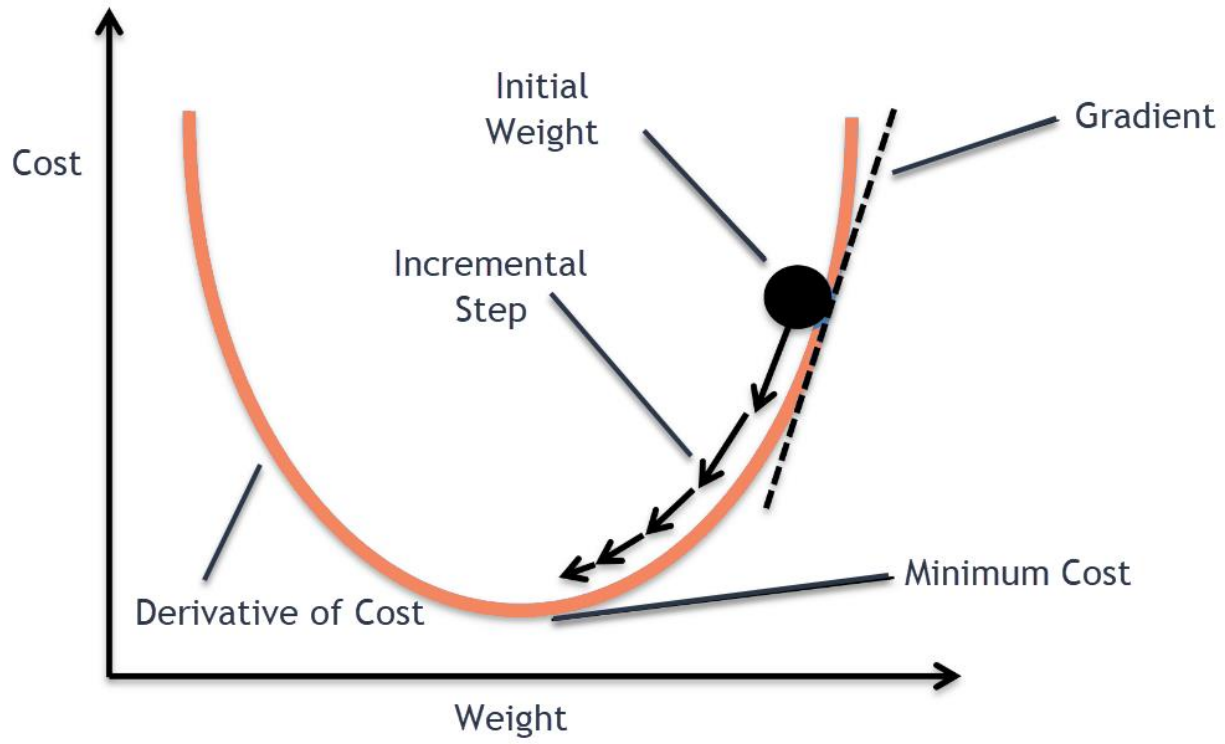

Figure 20: *Gradient Descent Algorithm [9]*

We choose to use SGD in the payload blast classification. To understand how SGD differs from the other two, we need to define what is meant by batch. Batch denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. The difference between Batch Gradient Descent and Mini-batch Gradient Descent is the batch size. In Batch Gradient Descent, the batch is taken to be the whole dataset. This very inefficient and computationally very expensive for a large dataset as is the case here. Mini-batch Gradient Descent is similar to Batch Gradient Descent, but the batch size is not necessarily the entire dataset. With SGD, only a single sample is passed through the neural network and the parameters of each layer are updated with the computed gradient. The sample is randomly selected from the dataset each iteration. Therefore, SGD makes it easier to fit

data into memory, which is a great concern in CNN. Moreover, SGD is computationally fast and the parameters converge faster as they are updated more frequently. What makes SGD an even a better choice than the other two is that it can help to get out of a local minimum due to the frequent updates [9]. It is however worth pointing out that the SGD takes a higher number of iterations to reach the minima because of its randomness in its descent. The mathematical description of SGD is as follows,

$$\theta_j \rightarrow \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

Where θ_j corresponds to the parameter to update, α is the learning rate (step size), and $J(\theta)$ is the loss function. In the next section, we investigate what has been achieved using CNN and the possibility of utilizing existing CNN models to train our classifier.

2.4. ImageNet

ImageNet is an annual competition with the intention to advance image classification tasks [10]. The goal is for the competitors to design a CNN that is capable of classifying a thousand classes with a dataset size of over 14 million images that have been hand annotated. The dataset includes images of dogs, cat, hat, coffee, to mention a few. Therefore, the goal is to classify these mixed images to their corresponding categories, classes. The model with the highest accuracy wins the competition. To emphasize how good CNN models have become at this task, it is worth looking at how they compare to human's classification abilities. As shown in Figure 21, 2015 Microsoft's ResNet model, the winner of the 2015 ImageNet competition, has a top-5 classification error of 3.6% vs. 5% for human. Top-5 error is the percentage of events classified where the top five predicated classes, of the trained 1000 classes, did not include the targeted class. This number has since improved, only to show why

artificial intelligence has grown to become a topic of interest once again. For comparison, the best performing model that existed before CNNs has a Top-5 error of 26%.

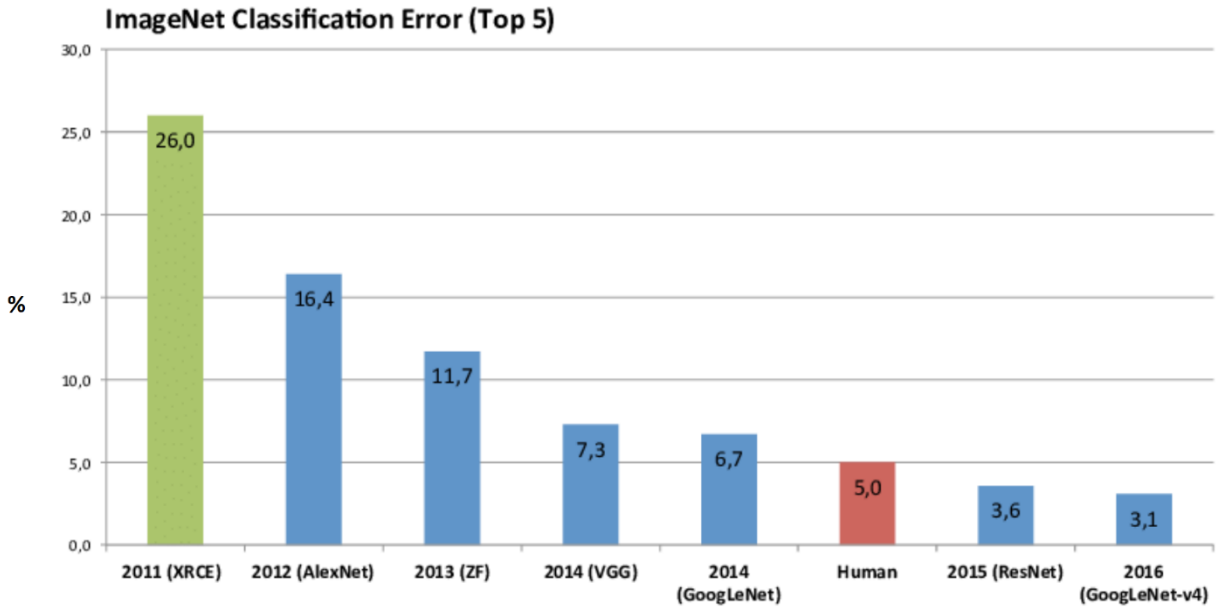

Figure 21: *Winner Results of ImageNet competition [11]*

This competition is significant as it pushes the boundary of what is possible using AI and for this specific research in that these pre-trained models can be used as a starting point even though the data set is significantly different and underrepresented in these models. To understand why pre-trained models are good starting points, one needs to understand what these models try to capture, namely features. The features of interest for our model are edges, being able to tell edges apart allows the model to easily locate features, see Figure 22.

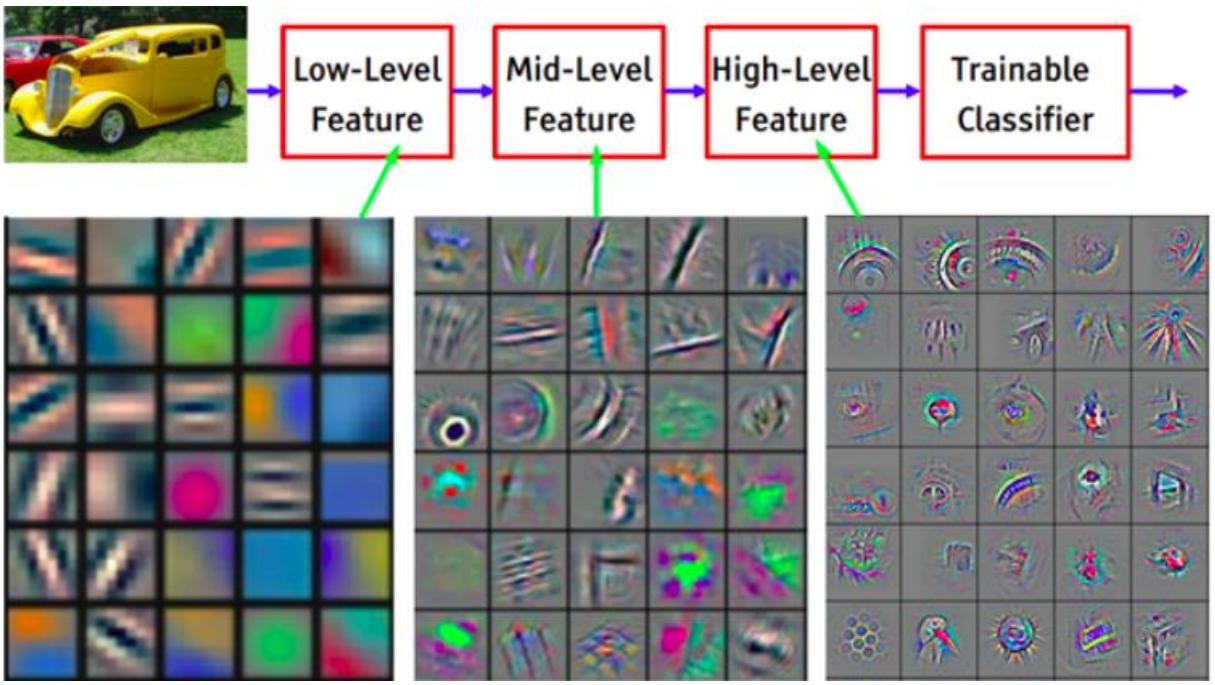

Figure 22: *Features map of a trained model extracted by evolving Convolution Layers [12]*

4.0. Methodology

The key to any good classifier is data and, in this case, a substantial amount of it- ideally over 100,000 events. It is fortunate that there is no lack of raw data, but the issue remains that the data is not categorized. Therefore, the first step is to manually classify the visual data of ANITA and determine which events are payload blasts. The goal here is to have a good sample dataset size to start with.

The next step of the process is to train a CNN model on the small amount of the data that was classified manually, around 80 events. The accuracy of this model did not exceed 50%, given how small the data sample size is. The accuracy here refers to the percentage of events that were accurately classified as payload blasts. Using this model made it easier to classify payload blasts and therefore increase the number of payload blasts sample from around 80 to over 1000. With a

few more similar runs, we were able to get the number of correctly classified payload events to over 16000. This allowed the process to move on to the next step.

To train a CNN model, we need to define a few parameters, first being the number of classes; there are two – payload events and non-payload events. Once the number of classes is known, the next step is to set up directories for each class. The convention is to use 50% of the data for training, 30% for testing, and 20% for validation, thereby making a total of three directories for each class. These percentages are not a hard requirement and can be adjusted for what is best for the model. The training dataset is used to train the model and it is where the model learns the features and adjust the parameters. After every training epoch, the model is validated using the validation dataset. This dataset is used to minimize overfitting by observing the accuracy of the model over the training dataset vs. the validation set. If the accuracy over the training set is much higher than that over the validation set, the model is overfitting and the training should stop or some randomness should be introduced to the model parameters. The testing dataset is used to assess the performance of a final model.

Creating a CNN has never been easier with libraries such as TensorFlow [1] by Google, and PyTorch [2], both of which happened to support Graphics Processing Unit (GPU) acceleration for optimal speed. I had the chance to play with all major libraries and I have successfully implemented the classifier on all. I was however faced with the question of whether to implement my CNN from the ground up or whether I should train a pre-trained model on the dataset. After experimenting with both, it has come clear to me that ANITA data is complex enough that it can utilize pre-trained models that utilize millions of parameters and therefore can capture a lot more information.

To understand why pre-trained models are such good starting points, it is important to understand feature extractions. When a model is trained on a dataset, the model attempts to catch the features that make the classification agree with expectation. The simplest of these features is edge detection. It allows the classifier to distinguish between features e.g., see Figure 23. The models that were selected for this research are indicated in Table 1 with the Top-1 and Top-5 error rate shown. The Top-1 error indicates the percentage of events where the top prediction is not the expected. Top-5 error, on the other hand, is the percentage of samples classified where the top five predicated classes do not include the targeted class, the class which the test sample actually belongs to. These models have been chosen to test whether different CNN architectures would impact the classification accuracy. Some of these models are extremely complex with over 138 million parameters to train as in the case of VGG-16, a CNN model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [14]. It is important to understand the yield of implementing a complex model and the downside. As the model grows larger, it utilizes more parameters, millions of them, which need to be trained. This also means that even testing the model can be inefficient and computationally expensive. Using complex models however increases the likelihood of overfitting.

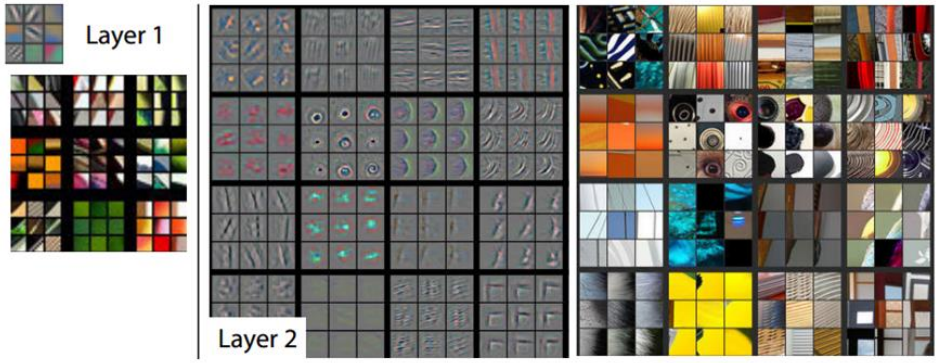

Figure 23: *Visualization of the layers of a trained CNN model [13]*

Network	Top-1 error	Top-5 error
AlexNet	43.45	20.91
Densenet-121	25.35	7.83
Inception v3	22.55	6.44
Squeezenet	41.90	19.58
VGG-11 with batch normalization	29.62	10.19
Resnet18	30.24	10.92

Table 1: *Top-1 error and Top-5 error as reported by ImageNet on the ImageNet dataset for various models [2].*

The process of utilizing pre-trained models is best understood by walking through the different layers in VGG-19 [14], an improved version of the CNN model, VGG-16 [14], mentioned earlier, see Figure 24. This network is 19 layers deep and combines the previously explained Convolution Layer, conv, Max Pooling Layers, maxpool, and two Fully Connected Layers, FC1 and FC2. The first layer takes an input of 224*224 and the different layers get applied to the input. Utilizing the libraries mentioned earlier, one can easily implement these models and make any suitable adjustment. The one of particular interest is the last Fully Connected Layer, FC2. This layer has a size of 1000, which is the number of classes this model is meant to be used on. For our use case, there are only two classes and therefore this layer is dropped and replaced with a Fully Connected Layer of size 2.

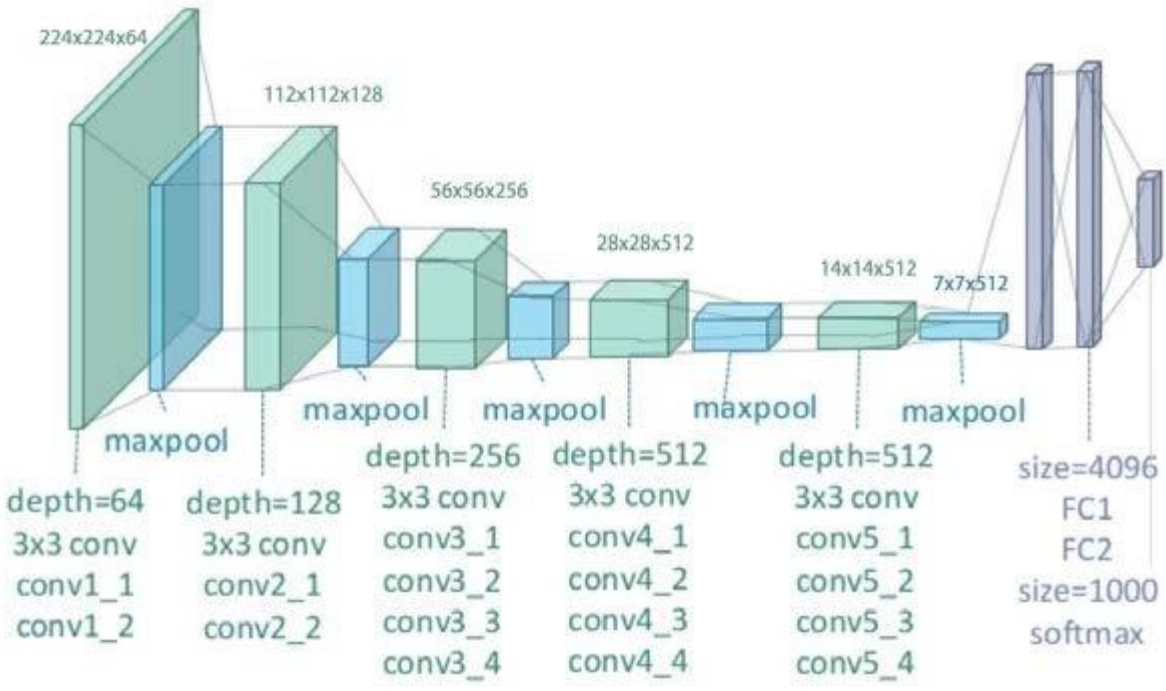

Figure 24: *VGG-19 Architecture [15]*

In this part of the process, one could try to use any of the pre-trained models to test its accuracy on our dataset only to find out how terrible it performs as none of the pre-trained models has been trained on our dataset. There are two ways to improve performance, we could either train the entire model, +130 M parameters, or just a portion of the model, the parameters that connect FC1 to FC2. From testing, both have shown comparable results indicating that the previous layers, Max Pooling and convolution layers, have captured good enough features to work on a class that has never seen.

In designing a CNN, there are many parameters to decide on. Some of these regards the architecture of the network, for example, the number of Convolution Layers, Fully Connected Layers, and Max Pooling Layers, etc. Furthermore, for the Convolution Layers, one needs to decide on the window size and stride size. Similar parameters need to be chosen for the Max Pooling Layer and Fully Connected Layer. This is easily solved by implementing different architecture; the models mentioned earlier have been shown through testing to work very well.

The other choice is the loss function and the optimizer. The optimizer chosen to train the models is the Stochastic Gradient Descent explained earlier. The loss of function of choice here is the Cross-Entropy Loss Function. From testing, this loss function yields the best results for the payload dataset. Cross-entropy loss, or log loss, measure the performance of a classification model whose output is a probability value between 0 and 1. A perfect model would have a log loss of 0, see Figure 25. In binary classification, as the case here, cross-entropy can be calculated as follow:

$$-(ylog(p) + (1 - y)\log(1 - p)) \qquad (1)$$

Where y is 1 if the class is predicted correctly or 0 otherwise, p is the predicted probability observation of the correct class.

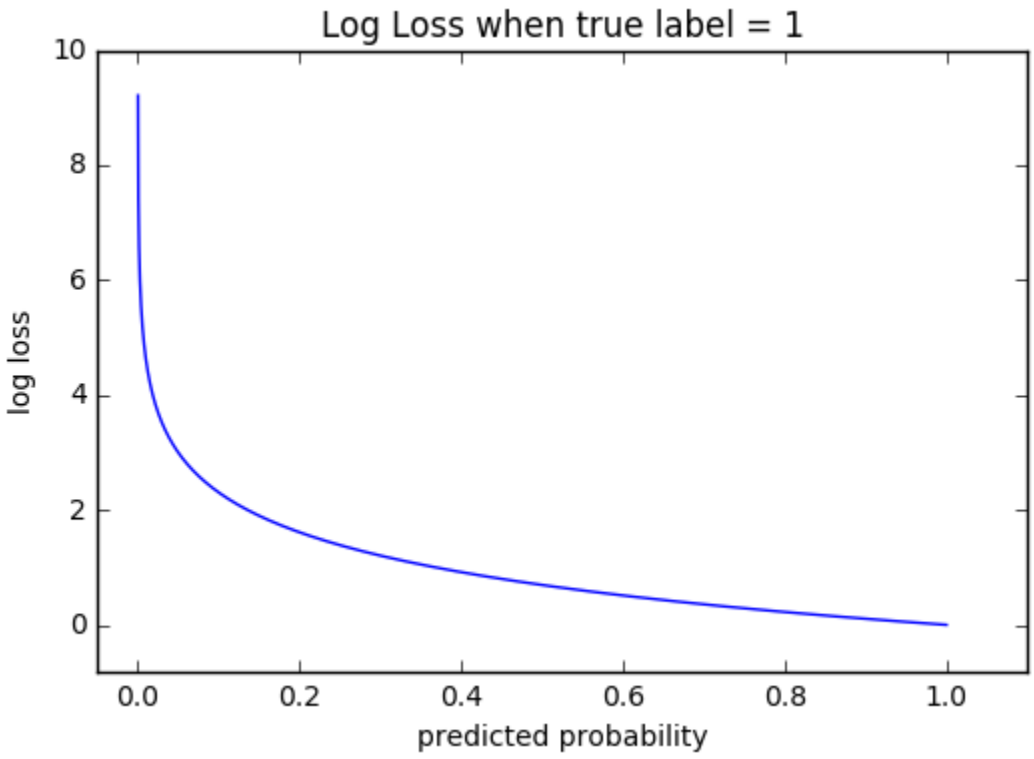

Figure 25: *Cross-Entropy Loss function*

5.0. Results

Table 2 shows the different models that have been trained on a dataset of size 5000 events and the accuracy of each model. The table includes four columns: validation accuracy, validation loss, training accuracy, and training loss. Validation accuracy refers to the percentage of events that have been classified correctly. This is the combined accuracy of payload blasts being classified as payload blasts and non-payload blasts events being classified as non-payload blasts. Validation loss relates to the loss function, the cross-entropy loss function in this case, which is a measurement of the distance between our prediction and the true value; the true value here is 1 if the sample is predicted correctly and 0 otherwise. Training accuracy and training loss also indicate the same thing but instead on the training dataset. From Table 1 it is clear that the data require somewhat of a complex model to accurately classify payloads events. At the same time, the accuracy does not scale linearly scale with the model complexity, VGG-11 vs. AlexNet. This is further demonstrated by Table 3 where only the parameter for the final Full Connected Layer is trained. Table 4 gives a good insight into the computation power required to train these models and how the model complexity affects the batch size that can be loaded on to the GPU memory and therefore the time it takes to train the models. The training results shown below are for 5 epochs, one epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. The models were trained on a system powered by an AMD Threadripper 1950X CPU with 16 Cores/32 Threads clocked to 4.0 GHz. The main computational power behind training the model is the GPU which utilizes CUDA acceleration, CUDA is a parallel computing platform created by Nvidia. The one used for the training is the NVDA RTX 2080 Ti, which is the highest-end consumer card available on market as of 2019. The system used to train the models is of a great significance because it shows the limiting factor in the training process, which from the table

3 is the GPU Memory. Although the RTX 2080 Ti integrates 11 GB of GDDR6, CNN is always limited by the memory capacity, which is shown by the maximum batch size possible for every model.

Network	Validation Accuracy	Validation Loss	Training Accuracy	Training Loss
AlexNet	97.25	0.0697	95.06	0.1304
Densenet-121	98.24	0.0463	98.02	0.0544
Inception v3	98.79	0.0439	98.14	0.08
Squeezenet	98.57	0.0636	96.16	0.1094
VGG-11 with batch normalization	98.35	0.0406	98.28	0.0478
Resnet18	97.80	0.0763	98.25	0.0656

Table 2: Final V*alidation accuracy/loss and training accuracy/loss for the trained models*

Network	Validation Accuracy (Update only FC2 parameters)	Validation Accuracy (Finetune the whole model)
AlexNet	97.2467	97.25
Densenet-121	94.1630	97.2467
Inception v3	95.1542	98.7885
Squeezenet	93.5022	98.5683
VGG-11 with batch normalization	94.0635	98.3480
Resnet18	93.0617	97.7974

Table 3: *Validation accuracy difference between training the entire models vs. the final layer parameters*

Network	Maximum Batch Size	Time	Validation Accuracy	Batch Size	Time	Validation Accuracy
AlexNet	1200	135	93.0617	100	87	97.2467
Densenet-121	600	130	86.3436	50	88	94.1630
Inception v3	750	135	90.0881	75	111	95.1542
Squeezenet	700	126	94.8238	200	91	93.5022
VGG-11 with batch normalization	150	93	93.017	75	89	94.6035
Resnet18	800	136	85.1322	200	92	94.0529

Table 4: *Model complexity as determined by the maximum batch size*

6.0. Conclusion

6.1. Summary

This project aim of classifying payload blasts has been to a great degree, a success. This is done through implementing different CNN architectures which were on average able to achieve an accuracy of around 98% by training them on our dataset. These payload blasts which we suspect to be interference by ANITA payload contribute around 1% of our data set and using the method described above can finally be removed.

6.2. Future Directions

Although 98% might seem high, the fact of the matter it is not sufficient. Ideally, we would like to reach an accuracy of around 100%. To get any closer to that, we require a lot of

training data. The accuracies reported above are based on a relatively small dataset of around 5000 events. The next step therefore is to use the existing model to grow the training dataset to build a better classifier that can achieve a near 100% accuracy. How close to a 100% accuracy we can get is based on the quality of the dataset and very hard to determine. In addition, it has been observed that using the models on a different run from the one which the training dataset is obtained from results in an accuracy that is worse. To avoid this, we could mix our training dataset to include events from different runs and potentially different ANITA flights. Furthermore, one can use the same models to remove other background events such as digitizer glitches by simply changing the dataset and retrain the models; the code has been written for that to be possible [16].

7.0. References

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer. Automatic differentiation in PyTorch. 2017.
3. A A Watson. High-Energy Cosmic Rays and the Greisen-Zatsepin-Kuzmin Effect. arXiv:1310.0325v1 [astro-ph.HE], October 2013.
4. P. W. Gorham, S. W. Barwick, J. J. Beatty, D. Z. Besson, W. R. Binns, C. Chen, P. Chen, J. M. Clem, A. Connolly, P. F. Dowkontt, M. A. DuVernois, R. C. Field, D. Goldstein, A. Goodhue, C. Hast, C. L. Hebert, S. Hoover, M. H. Israel, J. Kowalski, J. G. Learned, K. M. Liewer, J. T. Link, E. Lusczek, S. Matsuno, B. Mercurio, C. Miki, P. Miocinovic, J. Nam, C. J. Naudet, J. Ng, R. Nichol, K. Palladino, K. Reil, A. Romero-Wolf, M. Rosen, D. Saltzberg, D. Seckel, G. S. Varner, D. Walz, F. Wu.. Observations of the Askaryan Effect in Ice. arXiv:hep-ex/0611008v2, January 2017.
5. O. Banerjee. Studies in particle astrophysics with the ANITA experiment. 2018.
6. L. Serrano. A friendly introduction to Convolutional Neural Networks and Image Recognition. 2017.

7. X. Jin, & Cheng, C. Peng, W.C., H. Li. Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder. Physics of Fluids. 30. 10.1063/1.5024595, 2018.
8. I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.
9. D. Kapil. Stochastic vs Batch Gradient Descent. *Medium.com*, Jan 2019.
10. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei. mageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575v3 [cs.CV], Jan 2015.
11. Gustav v. Z. (2017). Survey of neural networks in autonomous driving.
12. H. Pokharna. The best explanation of Convolutional Neural Networks on the Internet!. *Medium.com*, Jul 2016.
13. A. Ananthram. Deep Learning For Beginners Using Transfer Learning In Keras. *towardsdatascience.com*, Oct 2018.
14. K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556v6 [cs.CV], Apr 2015.
15. Y. Zheng, C. Yang, A. Merkulov. Breast cancer screening using convolutional neural network and follow-up digital mammography. 4. 10.1117/12.2304564, 2018.
16. GitHub repository link to the code. https://github.com/abdualhag/Payload-blasts-classifier